Compiler Lec 10 notes

* Oral Exam of DB 6-1 at 10 AM
* Project Discussion ... Compiler

sec 1 and (sec 2 till no. 11) → 31-12 at 12 PM.

sec 2 from no. 12 and sec 3 ⇒ 2-1 at 1 PM

← زي ما ال (lexical) فيه حاجات مبيقدرش يعملها (identification)
فيروح لل (Parser) ← برده ال (Parser) فيه حاجات مبيقدرش
يتعامل معاها فيروح لل (symantic).

← ال (Phases) الأخيرة 3 ال (front end)

← Lexical, Parser, semantic → Page 4 on slide.

Semantic

← مسئول عن تحديد ال (scope) لل (Variables).

← في مراحل ال (front end) كنت بعمل (check) للغات
الموجوده معانا فقط.

← يعني اللي (Compiler) بيعمله لسه متكلمناش عنه.

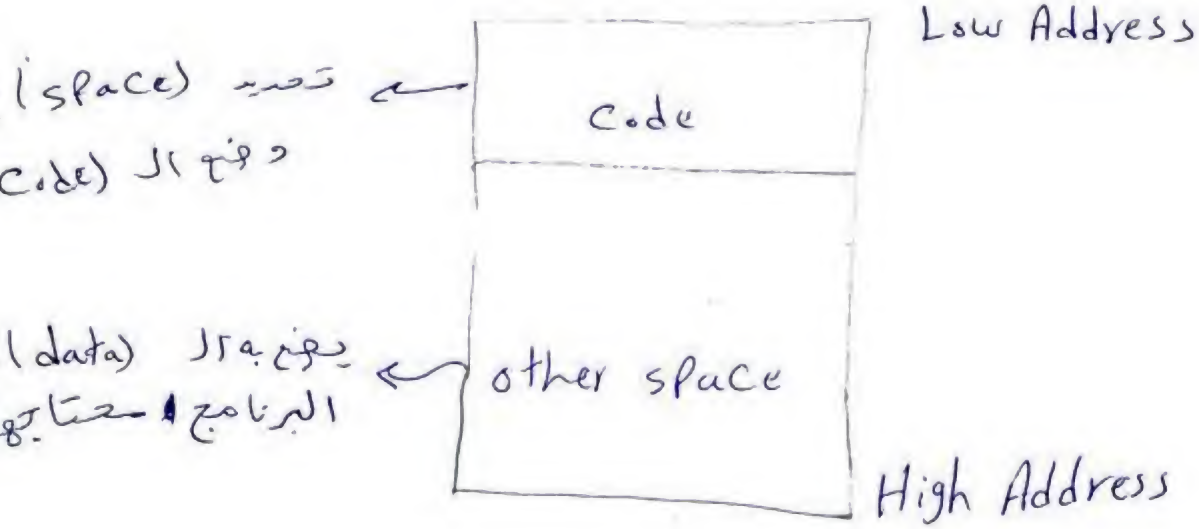
Run-time ← مقصود بيها ال (execution) ← dynamic

Notations

"OS" ← يعمل (memory allocation) عشان يعمل (space) للبرنامج.

~~(code) يعمل (code) ال~~ ←

Memory



← تحديد (space) يتم فيه وضع ال (code) بـ.

← يتم وضع ال (data) الى البرنامج محتاجها.

← شكل افتراضي لشكل ال (memory)

ال (Compiler) مسئول عن الأوركسترا بتاعة تنظيم استخدام ال (data) في البرنامج.

هو "روجر فيدرير" بتاع البرنامج

We need

1) our Program more efficient.

2) our Program is executed rapidly.

→ as fast as correct as possible

1) Execution is sequential ⇒ is this always happens?

← يوجد شيء (اسمه Parallel Computing) ممكن فجأة البرنامج عندنا يشتغل أكثر من (Processor) فيكون ال (Execution not sequential)

→ life time → is a dynamic Concept.

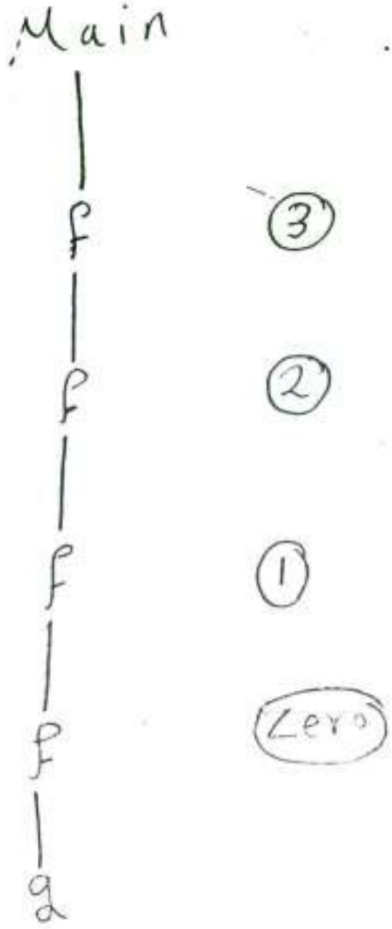
(run-time) يعني (execution) ال وقت ← عشان يشتغل

scope → static Concept

← ال (scope) هو الجزء اللي أنا شايف فيه ال (Variable)

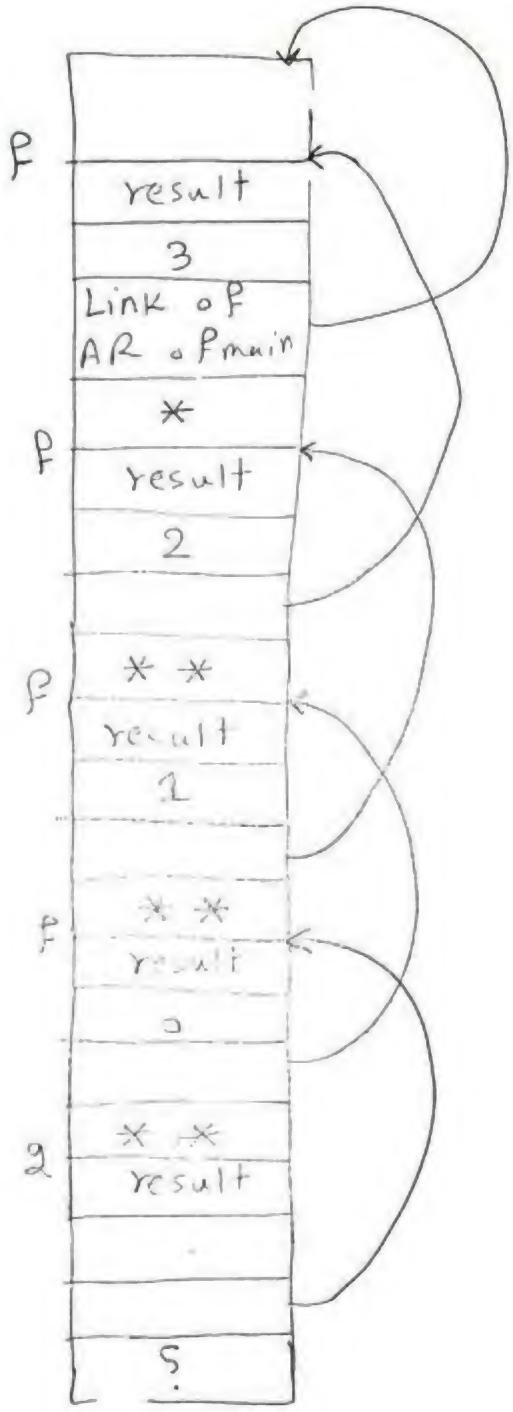
(Compile-time) يعني أنا بعد ،

Page 21



Stack Part

في الصفحة القادمة



← ممكن تيجي في الإستحانه.

ال (offset) ثابت.

← لو دافت عنه اول f ممكن نعمل (SKIP) لـ ٤ مراحل وناخد f التاليه.

5

Foo ⇒ create new object called Bar

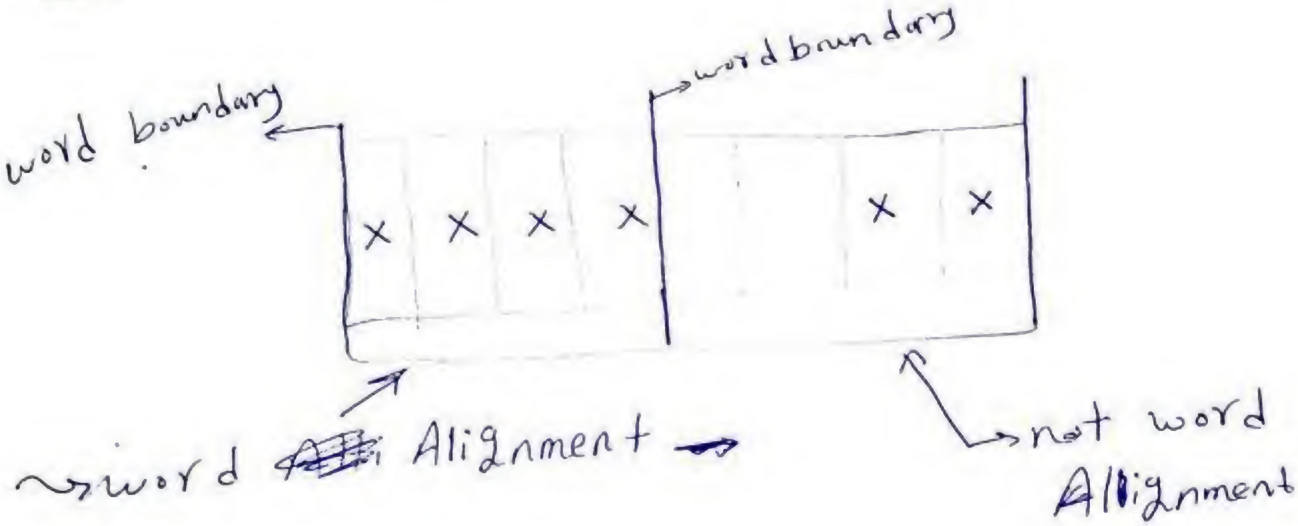
bar must be accessable by "foo"

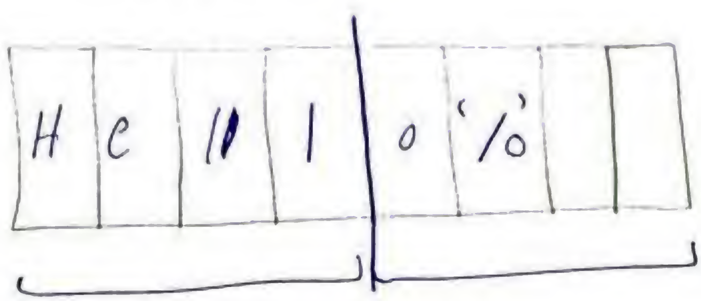
→ Page 42

← لا يحصل (crossing) بين stack) ، الجزء المحجوز ل (Heap

← معناها إنه مسافر في (memory space) والبرنامج يقف.

(OS) يقوم بعمل (Action) لحل المشكلة دي.





← هيعمل (skip) للجزء الباقي

Stack machine

$$r = F(a_1, a_2, \ldots, a_n)$$

أي (operation) هنفذها بعتمد على ال (stack) ←

Pop ← للي موجود في ال stack

Compute ← للعناصر دي

Push ← لل (result) دي تاني ل (top of stack)

$op(e_1, \ldots, e_n)$

sub expressions

① For each e_i $(1 < i < n)$

Compute e_i ← result in acc

[1→n-1] Push result on the stack

e_n ← Just evaluate (no need to Push)

② Pop n-1 values from the stack,
Compute op.

إنظر للتالي Page 54

③ store result in Acc.